PAL Programmable Logic Devices.

Consider implementing a general Boolean function of two variables A and B. This can be expressed as a sum of product terms.

The device shown in figure produces all possible (2^2) product terms formed by two variables. The inputs entered into the logic sum are selected by programmable fuses. Such devices are collectively termed as Programmable Logical Devices or PLDs.

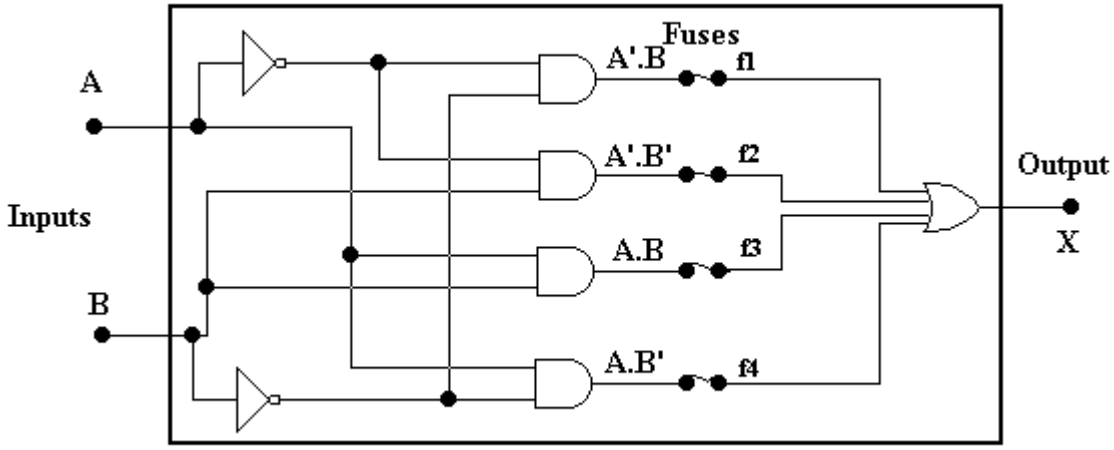

Figure 1

Example: X = A.B + A'.B' requires that fuses f1 and f4 to be "blown".

The above PLD has "fixed" AND gates and a "programmable" OR gate. If we wanted to implement m different functions, each having n inputs we would need a similar PLD with n inputs and m outputs. Inside the chip we would have 2^n AND gates and m OR gates. Each OR gate would have 2^n inputs, so the PLD would have to have a m x 2^{n+1} matrix of programmable fuses.

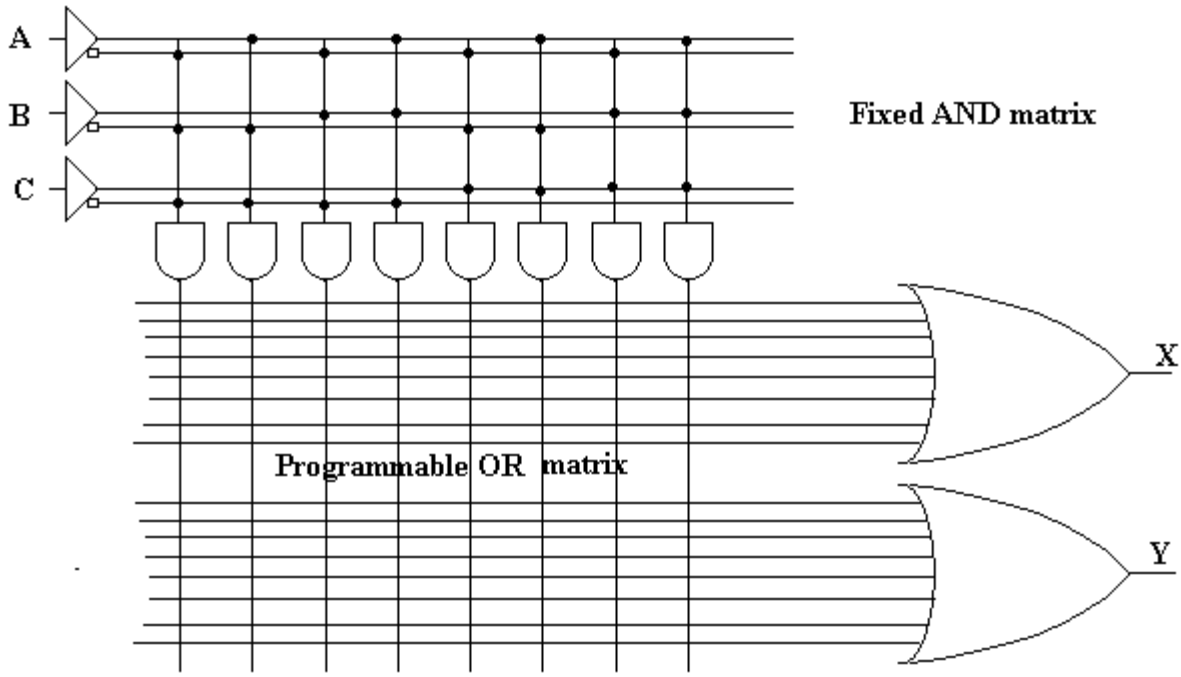

Figure 2 Simplified PROM

A simplified diagram of a 3 input x 2 output chip is shown in figure 2.
The 2 x 8 x 8 = 128 programmable fuses in the OR matrix are located at the intersections of the 16 horizontal with the 8 vertical lines. Each of the first 8 horizontal lines represents a possible minterm entering for the sum of products into the function X and similarly the lines 9 to 16 represent all the possible minterms entering into the sum for the function Y.
The PLD with fixed AND and programmable OR matrices is called PROM (Programmable Read Only Memory). It allows us to store m different truth tables, each having the same number of n variables.
PLDs are particularly useful for storing data. Imagine that we want to store 64 arbitrary 8 digit numbers. If we use a PROM with at least 5 inputs and 8 outputs, then each input combination (there are 2^5 =64) gives a different 8 digit number on the output. We can consider any given input combinations as an address in the memory were the eight digit number resides.

PAL (Programmable Array Logic).

Although great for data storage, PROMs are wasteful when used to represent logic functions. Most logic functions contain only a small number of minterms, whereas PROM provides us with all the minterms, whether we need them all not.
To implement a particular logic function it is more convenient to have a smaller number of product terms to choose from. But then we need the facility to "program in" those minterms which are actually contained in that function. This can be achieved using a PLD with programmable AND and fixed OR matrices. Such a PLD is known under the acronym PAL.

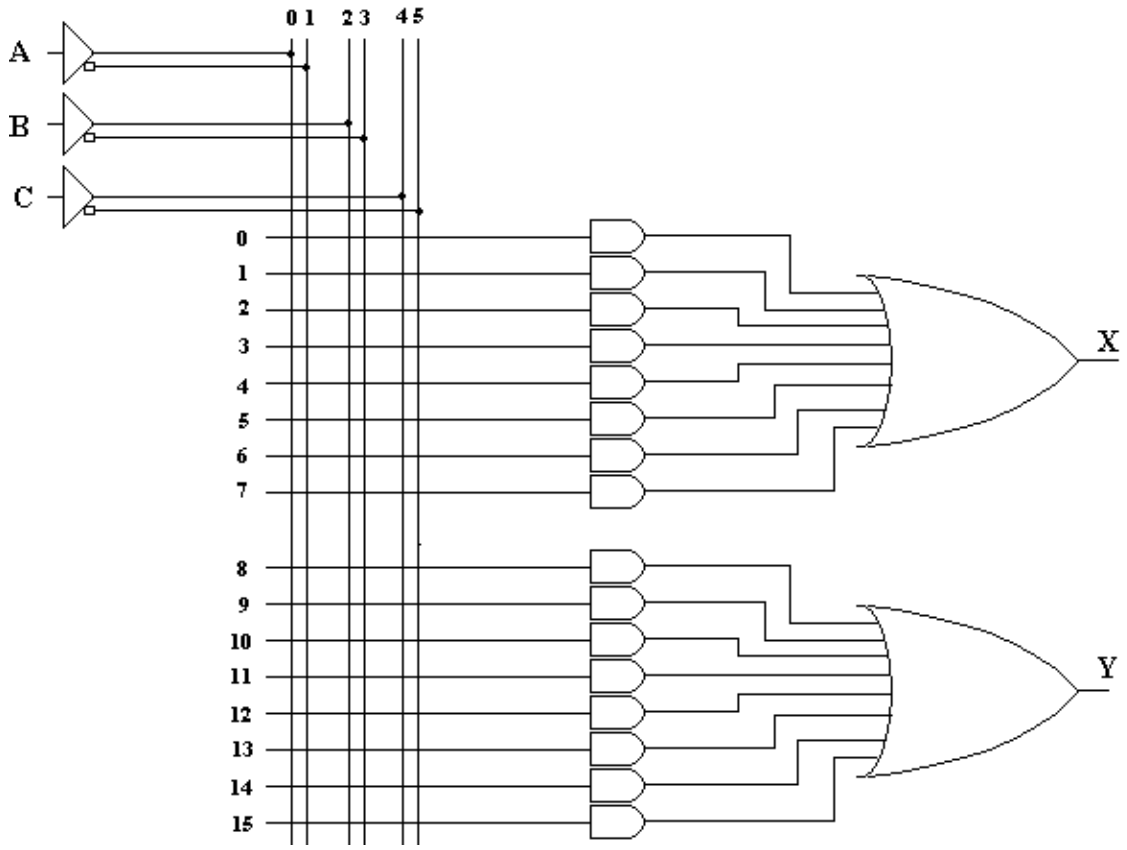

Figure 3: Structure of PAL

Returning to our example of two logic functions, each having 3 variables, consider how to implement on PAL the following two functions:

X = A.B.C' + A.B.'C + A'.B'.C'
Y = A'.B.C + A'.B'.C + A.B.C' + A.B.C

The first function requires 3 AND gates and the second four AND gates. Thus we need at least 7 AND gates and two OR gates.

The implementation using a 3 input x 2 output PAL is shown in figure 3.

The programmable AND array consists of six columns and 16 rows. Each intersection of a column and a row represents a programmable fuse. Rather than talking about the fuses we will picture an intersection where the fuse has not "blown" as a link.
Thus there are 6 x 16 = 96 programmable fuses (or links), somewhat smaller number than was needed for PROM.
If we denote a link by (row, column) it connects we see that we need the following links to implement the minterms in function X:

Minterm		Links
A.B.C'	~	(0,0) (0,2) and (0,5)
A.B'C	~	(1,0) (1,3) and (1,4)
A'.B'.C'	~	(2,1) (2,3) and (2,5)

Similarly for function Y.

Commercial PLDs come in several sizes. PAL chips have been around from early seventies so there is a "de facto" standard. The type of PAL is denoted as

PAL#inputs (designation letter) #outputs .

The most popular sizes are 16 inputs x 8 outputs (e.g. PAL16L8) and 20 inputs x 8 outputs (e.g. PAL20L8).

Consider PAL16L8. It has 16 inputs and 8 outputs. That is a total of 24 pins, adding the two power supply pins, we would expect a 26 pin package. Yet PAL16L8 is housed in a 20 pin package. This is possible because six pins can be configured to act either as outputs or inputs (they are bi-directional).

The figure 4 shows is the PAL shown in figure 3 with a modified pin X capable of providing input or output. The zero row minterm is used to program the tri state OR gate. If the OR gate output is enabled the pin is nominally an output pin, but it can be also used to provide internal feedback to the other OR gate. When the OR gate output is disabled the pin X can be used as an input pin.

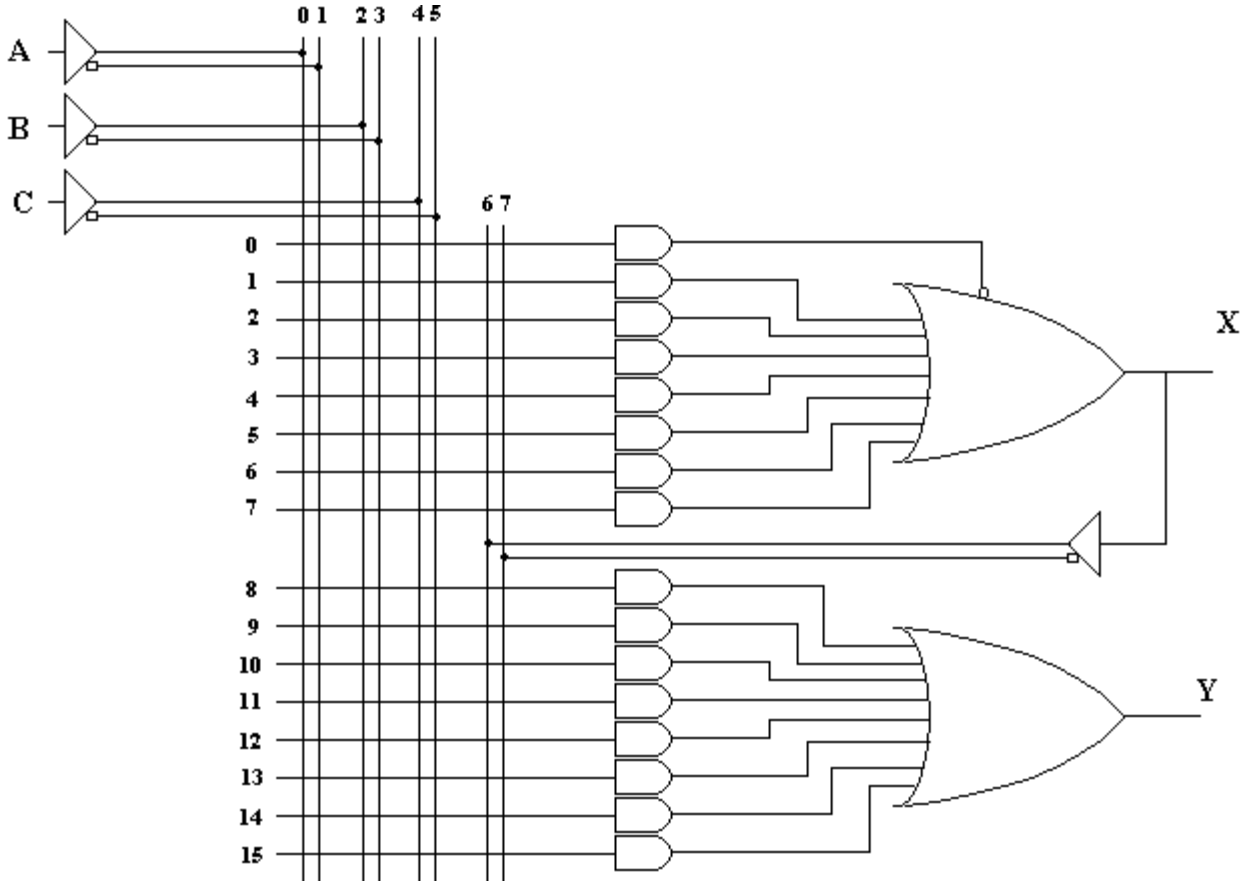

Figure 4: Structure of PAL with a combined input/output pin.

Figure 5 shows a detailed of another PAL modification allowing to programmatically invert the output pin. If the fuse on the input to the XOR is blown then the output of the OR gate will be inverted.

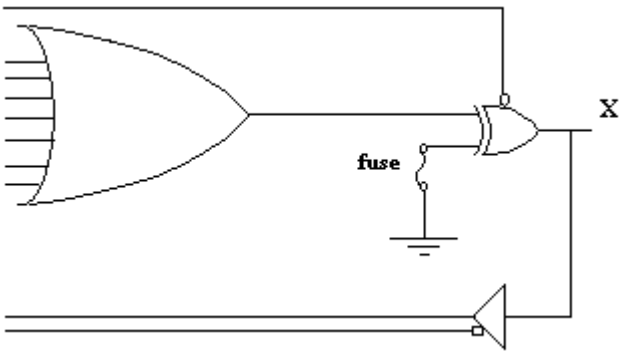

Figure 5: Programmatic selection of inverted output

Programming of PALs.

The programming file for PAL devices includes the so called fuse map and a set of test vectors. The fuse map shows which fuses have to be blown by the device programmer. The fuses that correspond to the nodes in the programming AND matrix are referred to by their “fuse address”, i.e. sequentially increasing numbers. The file also contains information on fuses that represent the internal configuration of the device, such as whether any particular OR gate has to have the output inverted or whether any feedback loops are present. The test vectors represent the expected outputs for a set of pre-defined inputs. Test vectors are used by the device programmer to test the PLD after it has been programmed. The format of the programming file has been standardized and is known as a JEDEC (Joint Electron Device Engineering Council) file.

Hardware Description Languages (HDL).

There are several HDLs that can be used to program PALs. The most commonly used are Abel, Verilog, VHDL and CUPL. In order to produce a downloadable JEDEC file, a source file using one of the above mentioned HDLs is compiled by an appropriate HDL compiler. Many PLD manufacturers produce CAD packages incorporating HDL compiler with additional utilities such as schematic capture software and functional and in-time simulation of the design.
Few examples of using the CUPL (Universal Compiler for Programmable Logic) are given below.

Example of CUPL Source for Logic Functions.

```
/* All comments are bracketed by /* and */ (similar to C source)                */
/* All CUPL statements terminate by a semicolon ;                               */
/* CUPL key words are not case sensitive, but the variable names are.           */

/******************************************************************************/
/* THIS PROGRAM DEMONSTRATES HOW CUPL COMPILES SINGLE GATES                     */
/******************************************************************************/

/*============================================================================= */
/*                           GAL 16L8      Chip Diagram                         */
/*============================================================================*/

/*                                 _______________                              */
/*                               |               |                              */
/*                        a x---|1             20|---x Vcc                      */
/*                        b x---|2             19|---x xnor                     */
/*                          x---|3             18|---x xor                      */
/*                           x---|4             17|---x nor                     */
/*                          x---|5             16|---x or                       */
/*                          x---|6             15|---x nand                     */
/*                          x---|7             14|---x and                      */
/*                          x---|8             13|---x invb                     */
/*                          x---|9             12|---x inva                     */
/*                       GND x---|10            11|---x                         */
/*                               |_______________|                              */

/* The device has ten dedicated inputs (pins 1 to 9 and 11), pins 13 through 18  */
/* can be configured either as inputs or outputs. Pins 12 and 19 can be only     */
/* outputs. All outputs can be switched into a high impedance by the product     */
```

```
/* term on the eighth OR gate input. The outputs of all OR gates can be inverted */
/* by blowing a corresponding internal configuration fuse.                        */
/* Note: In this example pins 1 and 2 are the inputs and pins 12 to 19 outputs    */
/* The true source code starts here:                                              */
/* The Mandatory Header has to be included in every CUPL source file              */

Company         Logical Devices, Inc.; /* does not matter what you enter here    */
Location        None;                   /* does not matter what you enter here    */
Assembly        None;                   /* does not matter what you enter here    */
Device          16L8a;                /* Important info:  PAL 16L8 device       */



/* Inputs:  define inputs to build simple gates from  two input variables a and b.*/
/* PIN is a CUPL keyword which assigns the variable to an IC pin                  */

Pin 1 =  a;  /* First input variable as pin 1 is dedicated to input */
Pin 2 =  b;  /* Second input variable as pin 2 is dedicated as an input*/


/* Outputs:  Define the bidirectional pins as outputs  active HI levels.          */
/*           That these pins are outputs is not yet obvious but they are          */
/*           implicitly defined because, as we shall see, they will appear        */
/*           on the Left HS of logic equations which determine their values.      */
*/

Pin 12 = inva;       /*  a'    */
Pin 13 = invb;       /*  b'    */
Pin 14 = and;        /*  a.b   */
Pin 15 = nand;       /* (a.b)' */
Pin 16 = or;         /* (a+b)  */
Pin 17 = nor;        /* (a+b)' */
Pin 18 = xor;        /* (a⊕b)  */
Pin 19 = xnor;       /* (a⊕b)' */


/*Now follows the Logic:  examples of simple gates expressed in CUPL        */
equation statements using CUPL specific BOOLEAN operators                   */
/* Left hand side  is output <=  Right hand side is input                    */

inva = !a;              /* inverters */
invb = !b;
and  = a & b;           /* and gate */
nand = !(a & b);        /* nand gate */
or   = a # b;           /* or gate */
nor  = !(a # b);        /* nor gate */
xor  = a $ b;           /* exclusive or gate */
xnor = !(a $ b);        /* exclusive nor gate */
```

As a by-product of compilation the WINCUPL compiler also produces a document file providing the information on the way the design is implemented. Below given are few extracts such as the Symbol Table and the Fuse Map appropriate to the above source code with the explanations as appropriate.

```
================================================================================
                                Symbol Table
================================================================================


Pin Variable                                        Pterms   Max     Min
Pol   Name              Ext     Pin     Type        Used   Pterms  Level
--- --------            ---     ---     ----        ------ ------  -----

    a                           1        V          -      -       -
    and                         14       V          1      8       1
    b                           2        V          -      -       -
    inva                        12       V          1      8       1
    invb                        13       V          1      8       1
    nand                        15       V          1      8       1
    nor                         17       V          2      8       1
    or                          16       V          2      8       1
    xnor                        19       V          2      8       1
    xor                         18       V          2      8       1
```

LEGEND: v = variable

Note: Each OR gate of the 18L6 can take of up to 8 product terms (AND gates). The above table shows the utilization of the internal gates.

```
================================================================================
                                Fuse Plot (see the fuse map shown below)
================================================================================
Explanations:

   1. Each row represents the status of fuses connected to the input of a single
      AND gate. Thus the number of rows under each pin gives the number of AND
      gates for each output OR gate. For example PIN #19 has eight rows, i.e. the
      output NOR gate has eight inputs, one from each of the eight AND gates.
   2. Each AND gate has to take a pair of direct and inverted inputs, so it has
      twice as many inputs as there are input pins on the chip. E.G. 16V8 has up to
      16 input pins so all AND gates can have up to 32 inputs.
   3. Each input literal (variable and its inverted value) is represented by a
      single columns. For the 16L8  there are 16 variables, thus 32 literals and
      therefore 32 columns. In our case we use only two input variables: a (pin 1)
      and b(pin 2). The first column represents the literal a, the second column
      its inverted value a', the third one b and the fourth column b'. The other
      columns represent the other input pins which are not used in this example.
   4. The intersections of columns and rows are termed the AND Programming Matrix.
   5. Each node of this matrix represents a single fuse which connects the literal
      represented by the corresponding column to the AND gate input represented by
      the row. A X signifies fuse not blown (that literal is connected to input of
      the AND gate) and a - fuse blown (not connected). So in our example the first
      row that represents the single AND gate has only the a and b' inputs
      connected. The second row that represents the second AND gate has only the
      inputs a' and b connected. The remaining six AND gates connected to output
      pin 19 NOR gate are not used. Thus there are only two minterm inputs to the
      OR gate associated with the output pin 19: a.b' ,  a'.b.
   6. Each row that starts with an output pin number provides internal
      configuration for the output "macro cell". In the example below the important
      data is the polarity of the output, designated by the POL fuse. If the
      polarity bit fuse is left connected the output polarity of the OR gate is
      inverted giving an NOR gate. Otherwise it is unaffected. In our example the
      fuse is intact so the gate at pin 19 will be inverting (NOR gate). The output
      of this pin will thus be : (a.b' + a'.b)' which is the formula for an
      inverting exlusive OR ( XNOR ). Similarly, the output at pin 18 (not
      inverting as fuse is blown) yields a.b' + a'.b and that is a XOR function.
   7. The fuses are labelled by sequential numbers known as "fuse addresses". The
      left hand side column gives the starting address of the first fuse in each
      row. For example, the AND matrix corresponding to the output pin 19 has fuses
```

from address 0000 to 00255. In our example only the fuses with addresses 0, 3, 33 and 34 are connected.

8. The fuse map information is contained in a JEDEC file which (alongside other data) holds a table of fuse addresses and their state (intact or blown).

```
Syn   02192 - Ac0   02193 x

Pin #19  02048  Pol x  02120  Ac1 x
 00000 x--x----------------------------
 00032 -xx----------------------------
 00064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18  02049  Pol -  02121  Ac1 x
 00256 x--x----------------------------
 00288 -xx----------------------------
 00320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17  02050  Pol x  02122  Ac1 x
 00512 --x----------------------------
 00544 x-----------------------------
 00576 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00736 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16  02051  Pol -  02123  Ac1 x
 00768 --x----------------------------
 00800 x-----------------------------
 00832 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00896 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00960 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 00992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #15  02052  Pol x  02124  Ac1 x
 01024 x-x----------------------------
 01056 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01088 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01120 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01152 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14  02053  Pol -  02125  Ac1 x
 01280 x-x----------------------------
 01312 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01344 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01376 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13  02054  Pol -  02126  Ac1 x
 01536 -x----------------------------
 01568 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
 01600 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12  02055  Pol -  02127  Ac1 x
 01792 ---x----------------------------
 01824 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01856 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 01984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 02016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Example of CUPL Source for Table Entry.

To enter a table of (say) three inputs and six outputs we can use CUPL syntax TABLE.

First, define the input and output variable vectors using the syntax of **FIELD** and then use the syntax of **TABLE** to assign the values:

```
/* Inputs */
PIN  1 = a;
PIN  2 = b;
PIN  3 = c;

/* Outputs */
PIN  12 = u;
PIN  13 = v;
PIN  14 = w;
PIN  15 = x;
PIN  16 = y;
PIN  17 = z;

/* Define multi-bit variables for the input and the output */
FIELD my_inputs  [ a, b, c];
FIELD my_outputs [u, v, w, x, y, z ];

/* Assign output values to input values */
TABLE  my_inputs => my_outputs {

[0, 0, 0 ] => [ 1, 0, 1, 1, 0, 0];
[0, 0, 1 ] => [ 1, 1, 1, 1, 0, 1];
[0, 1, 0 ] => [ 1, 1, 1, 1, 0, 0];
[0, 1, 1 ] => [ 1, 0, 1, 1, 0, 0];
[1, 0, 0 ] => [ 1, 0, 0, 0, 0, 0];
/* and more entries if required */
}
```

PALs for Sequential Logic Design.

To implement sequential circuits the so called registered PALs are available. These have the output OR gates connected to internal flip flops. For example the PAL 16R8 is the registered version of the logic PAL 16L8. The main difference is that the six bidirectional outputs of 16L8 are routed through six separate D-flip flops. The clock input is pre-assigned to a specific input pin (pin 1 on 16R6) that cannot be used for other purposes.

GAL 16V8 - OMCL

Logic and registered PALs are job specific PLDs and are normally OTPs (one time programmable) only. The so called GAL devices feature CMOS re-programmable fuses so there is no practical limitation on the number of times they can be re-programmed. In addition, the GAL devices have another very useful feature. They can be programmatically configured to behave as PALs of various degrees of complexity. This is achieved by them having the so called Output Logic Macro Cell (OMCL) that can be configured to one of the three device modes: simple, complex and registered.
An example of a OMCL used is shown above. The configuration of the OMCL is determined by the fuses that control the inputs to the three multiplexers. In simple and complex modes 16V8 acts as a logic PAL by-passing the D flip flops. The simple mode allows no loop back. In registered mode 16V8 behaves as a registered PAL 16R8.

Explanations of some of the PLD Acronyms.

JEDEC File	A standardized format file containing the fuse map. The file is downloaded to the PLD during the programming cycle.
OTP	One Time Programmable PLD. The fuse map once programmed cannot be changed.
EP PLD	Electrically Programmable PLD. The fuse map can be deleted by exposing the PLD to UV light. The PLD then can be re-programmed.
EEP PLD	Electrically Erasable Programmable PLD. The fuse map can be deleted electrically. The PLD then can be re-programmed.
Register PAL	A PAL (such as 18R) where the outputs of OR gates are connected to D flip flops. Register PALs can be used for sequential logic design.
OLMC	The Output Logic Macro Cell. An output sub-circuit of a PAL which can be programmed to act either as a combinatorial or register PAL.
GAL	Generic Array Logic, the term used for EEP PAL having OLMC outputs.
CPLD	Complex PLD. Several hundred PLDs on a single chip with programmable interconnections.
FPGA	Field Programmable Gate Arrays. Contains a very large number of EEP cells. Each cell can typically contain a small PROM, a register and some gates. The interconnections between cells are also programmable.